GenArch (Generative Architectures): A Model-Based Product Derivation Tool

Elder Cirilo, Uirá Kulesza, Carlos Lucena

Software Engineering Laboratory
Computer Science Department
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)

Motivation

- Software Product Line (SPL) approaches motivate the definition of a flexible and adaptable architecture which addresses the common and variable SPL features;
- SPL architectures are implemented by defining or reusing a set of different artifacts, such as OO frameworks and software libraries;
- Recently, new programming techniques have been explored to modularize the SPL features, such as, aspect-oriented programming, feature-oriented programming and code generation.

Motivation

- Product Derivation refers to the process of constructing a product from the set of assets specified or implemented for a SPL;

- Over the last years, instantiation/derivation tools have been proposed to facilitate the selection, composition and configuration of SPL code assets and their respective variabilities;

- Examples of tools:
  - Gears
  - Pure::variants

Problem

- These tools are in general complex and heavyweight to be used by the mainstream developer community.

- Some problems/deficiencies from the existing tools:
  - they incorporate a lot of new concepts from the SPL development area;
  - definition of many complex models and/or functionalities;
  - they are in general more adequate to work with proactive approaches.

Our work

- This work proposes GenArch, a model-driven product derivation tool.

- It is centered on the definition of three models:

  (i) Feature model

  (ii) Architecture model

  (iii) Configuration model

- Our approach motivates:

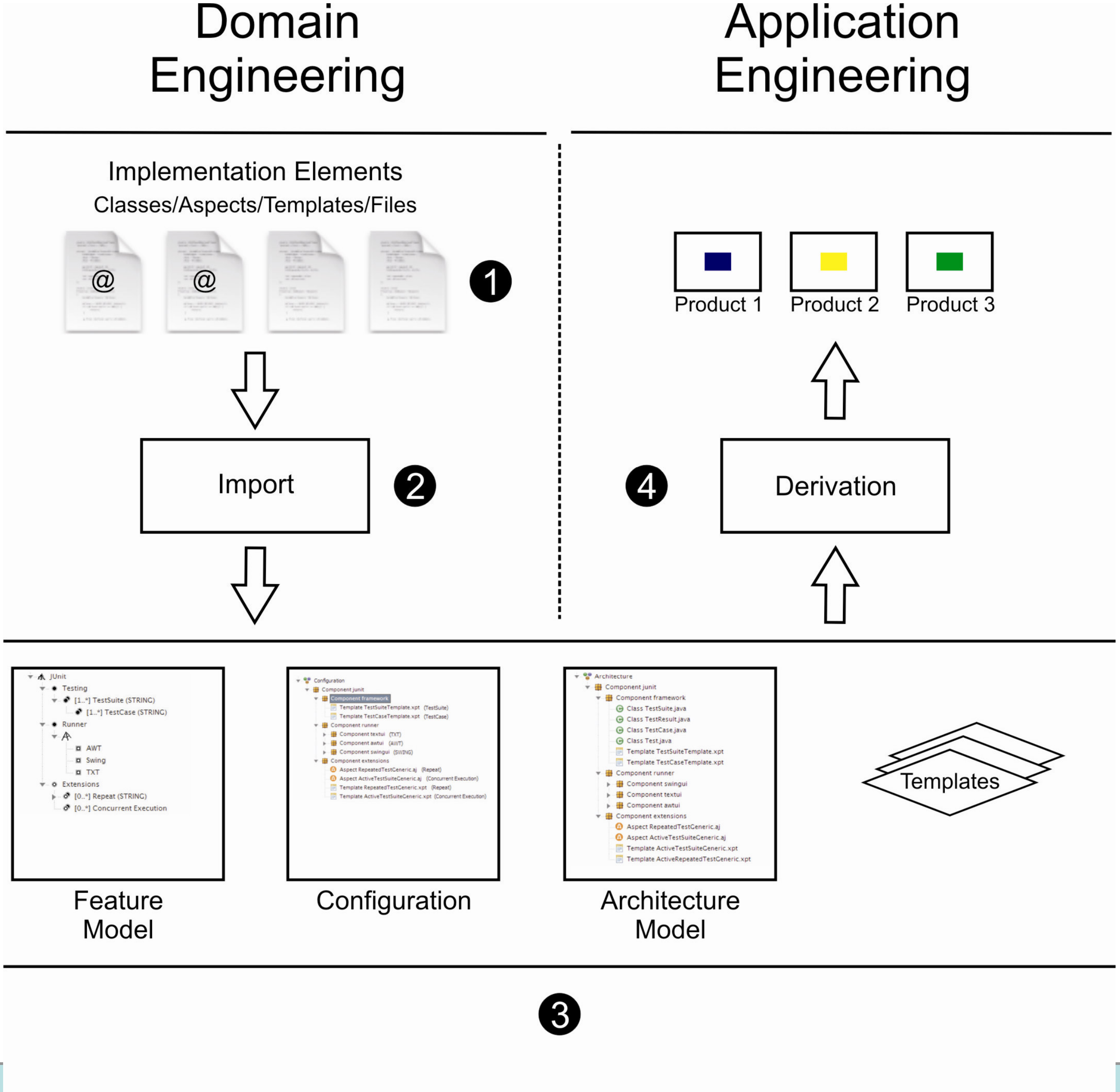
  - the generation of initial versions of these models based on a set of code annotations;

  - the refinement and adaptation of these initial versions to enable the automatic product derivation.

Agenda

- Introduction / Motivation
- Approach Overview
- Approach in Action
- Tool Architecture / Adopted Technologies
- Discussion and Lessons Learned
- Conclusions and Future Work

Approach Overview



Domain Engineering

Application Engineering

Implementation Elements
Classes/Aspects/Templates/Files

```
JUnit
  Testing
    [1..*] TestSuite (STRING)
      [1..*] TestCase (STRING)
  Runner
    AWT
    Swing
    TXT
  Extensions
    [0..*] Repeat (STRING)
    [0..*] Concurrent Execution
```

Feature Model

```
Configuration
  Component junit
    Component framework
      Template TestSuiteTemplate.xpt  (TestSuite)
      Template TestCaseTemplate.xpt  (TestCase)
    Component runner
      Component textui  (TXT)
      Component awtui  (AWT)
      Component swingui  (SWING)
    Component extensions
      Aspect RepeatedTestGeneric.aj  (Repeat)
      Aspect ActiveTestSuiteGeneric.aj  (Concurrent Execution)
      Template RepeatedTestGeneric.xpt  (Repeat)
      Template ActiveTestSuiteGeneric.xpt  (Concurrent Execution)
```

Configuration

```
Architecture
  Component junit
    Component framework
      Class TestSuite.java
      Class TestResult.java
      Class TestCase.java
      Class Test.java
      Template TestSuiteTemplate.xpt
      Template TestCaseTemplate.xpt
    Component runner
      Component swingui
      Component textui
      Component awtui
    Component extensions
      Aspect RepeatedTestGeneric.aj
      Aspect ActiveTestSuiteGeneric.aj
      Template ActiveTestSuiteGeneric.xpt
      Template ActiveRepeatedTestGeneric.xpt
```

Architecture Model

Approach Overview

- The purpose of each model of our approach

- Feature Model
  - Represent variabilities from the SPL architecture.
- Architecture Model
  - Offer a visual representation of code artifacts from the SPL architecture.
- Configuration Model
  - Define the mapping between features and code artifacts. It represents the configuration knowledge from a generative approach.

Agenda

- Introduction / Motivation
- Approach Overview
- Approach in Action
- Tool Architecture / Adopted Technologies
- Discussion and Lessons Learned
- Conclusions and Future Work

Approach in Action

- Illustrate the tool functionalities through an example.

- Approach Steps:

  I. Annotating Java Code with Feature and Variabilities

  II. Generating and Refining the Approach Models

  III. Implementing Variabilities with Templates

  IV. Generating SPL Instances

Framework JUnit

- Specification of unit and integration tests.

- Implementation of Variabilities:
  - Framework OO > polimorphism
  - Aspect-Oriented Programming

- Existing variabilities:
  - Test suites and test cases
  - Graphical User Interface (Swing, AWT, Txt)
  - Test cases extensions (repetition, concurrent execution) >> Aspects

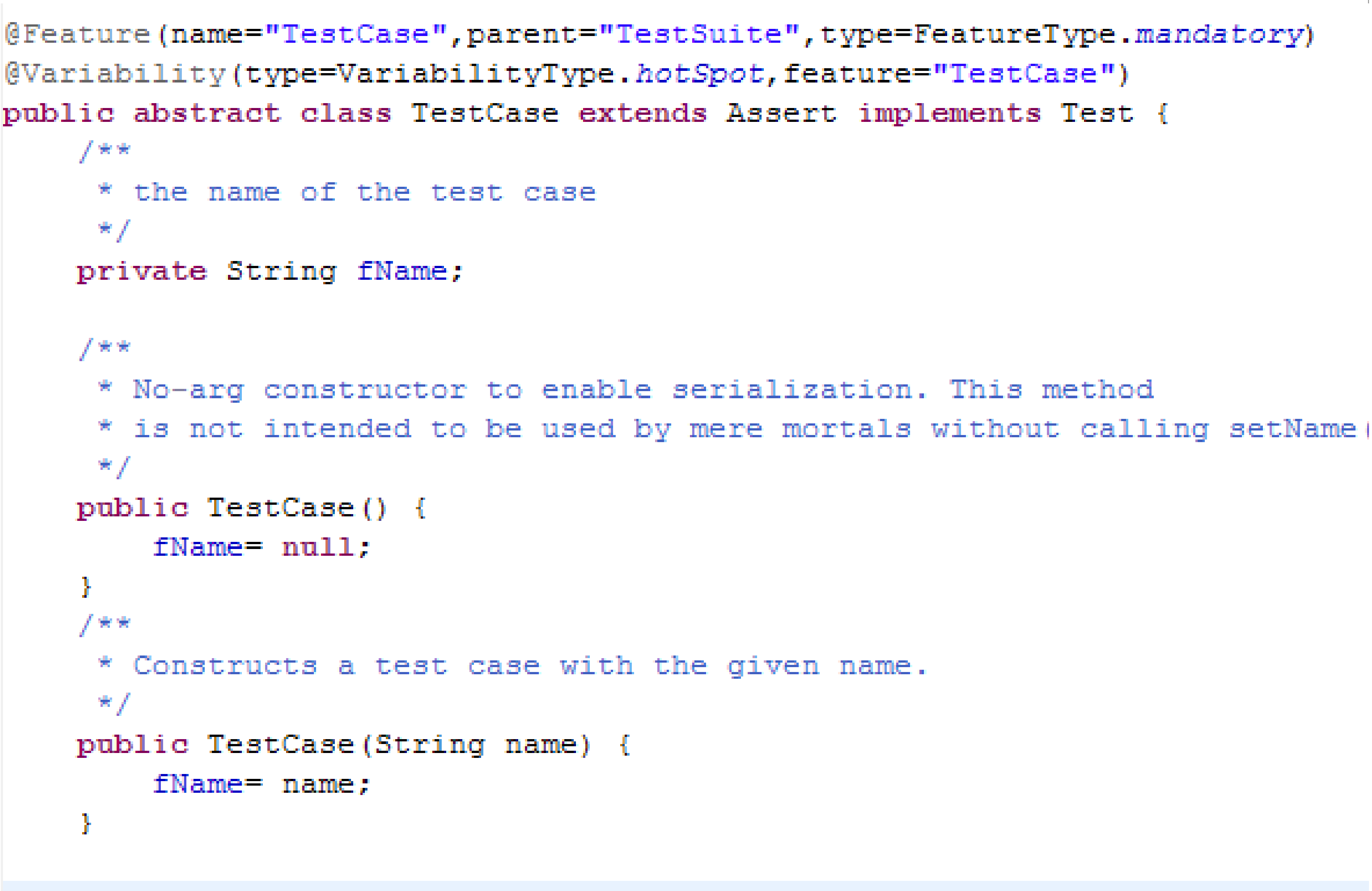
I. Annotating Java Code with Feature and Variabilities

- Two kinds of annotations: @Feature e @Variability
- Examples:

```
@Feature(name="TestCase",
            parent="TestSuite",
            type=FeatureType.mandatory)
@Variability(type=VariabilityType.hotSpot,
               feature=“TestCase”)
```

- They are processed by a parser to generate initial versions of the models

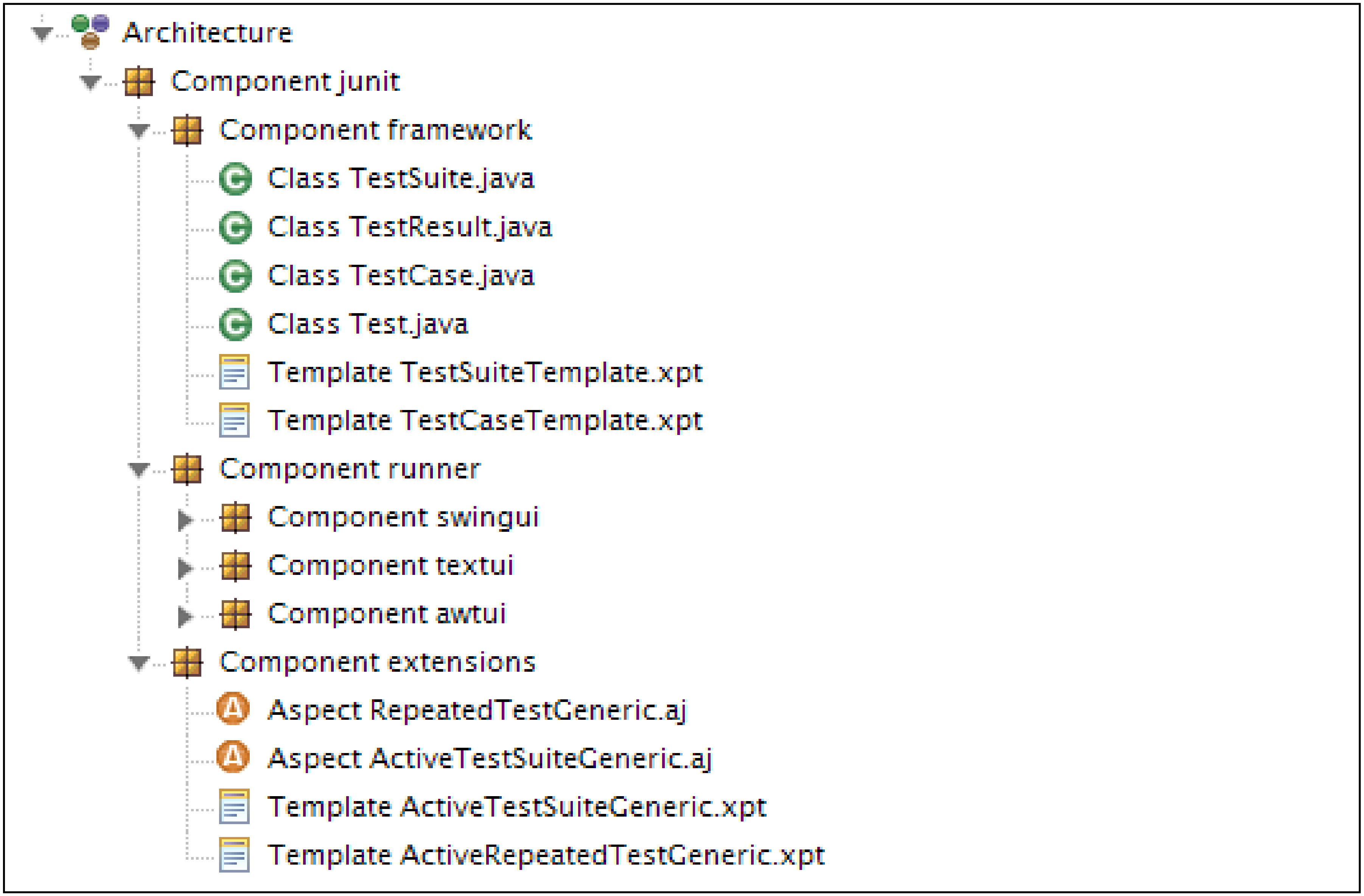
Example: TestCase class annotated



```
@Feature(name="TestCase",parent="TestSuite",type=FeatureType.mandatory)
@Variability(type=VariabilityType.hotSpot,feature="TestCase")
public abstract class TestCase extends Assert implements Test {
    /**
     * the name of the test case
     */
    private String fName;

    /**
     * No-arg constructor to enable serialization. This method
     * is not intended to be used by mere mortals without calling setName(
     */
    public TestCase() {
        fName= null;
    }
    /**
     * Constructs a test case with the given name.
     */
    public TestCase(String name) {
        fName= name;
    }
```

II. Generating and Refining the Approach Models

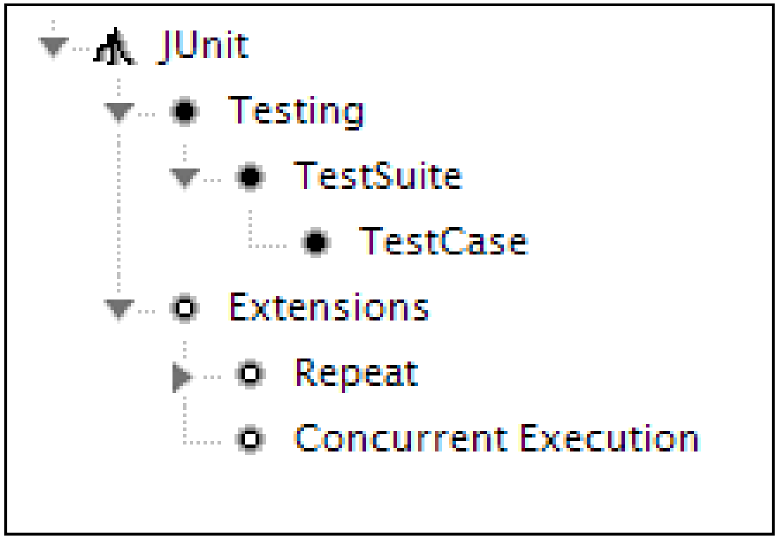


```
Architecture
  Component junit
    Component framework
      Class TestSuite.java
      Class TestResult.java
      Class TestCase.java
      Class Test.java
      Template TestSuiteTemplate.xpt
      Template TestCaseTemplate.xpt
    Component runner
      Component swingui
      Component textui
      Component awtui
    Component extensions
      Aspect RepeatedTestGeneric.aj
      Aspect ActiveTestSuiteGeneric.aj
      Template ActiveTestSuiteGeneric.xpt
      Template ActiveRepeatedTestGeneric.xpt
```

Laboratório de Engenharia de Software

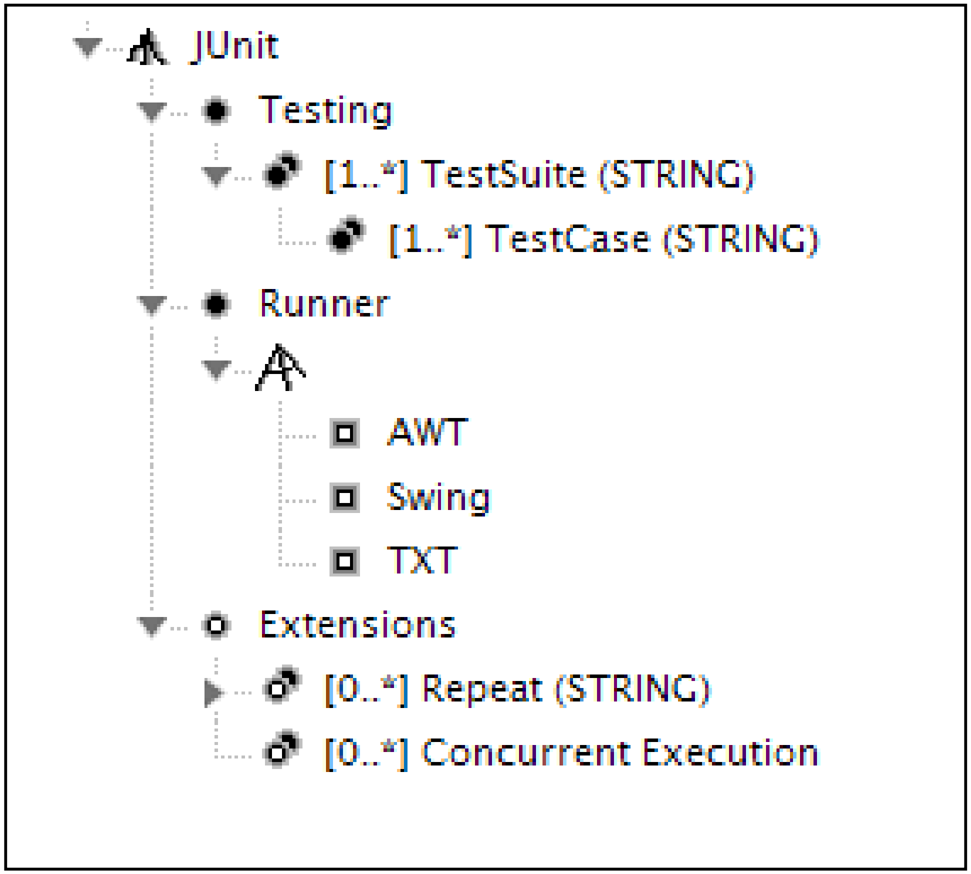
II. Generating and Refining the Approach Models

Laboratório de Engenharia de Software



- JUnit
  - Testing
    - TestSuite
      - TestCase
  - Extensions
    - Repeat
    - Concurrent Execution

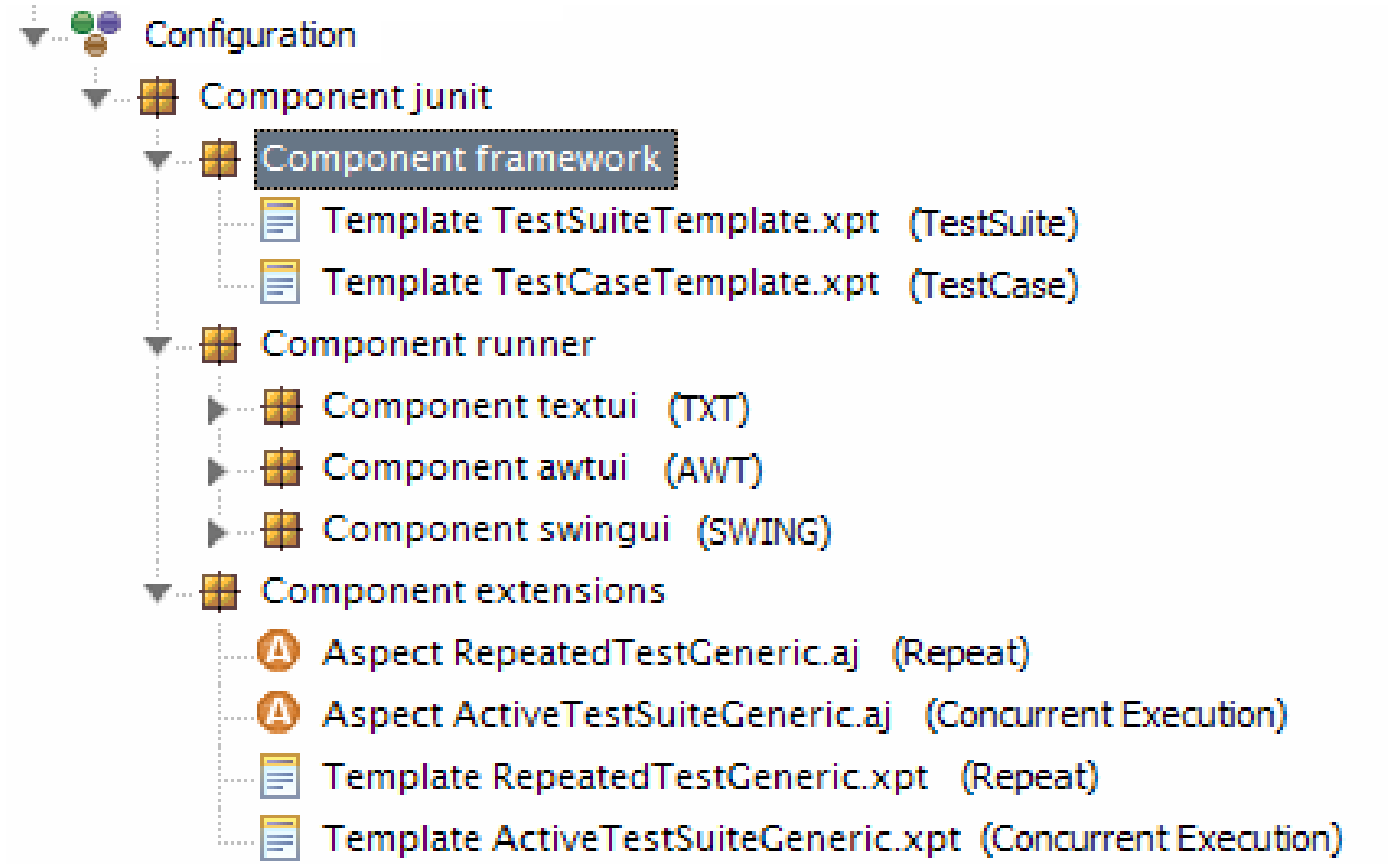
Before



- JUnit
  - Testing
    - [1..*] TestSuite (STRING)
      - [1..*] TestCase (STRING)
  - Runner
    - AWT
    - Swing
    - TXT
  - Extensions
    - [0..*] Repeat (STRING)
    - [0..*] Concurrent Execution

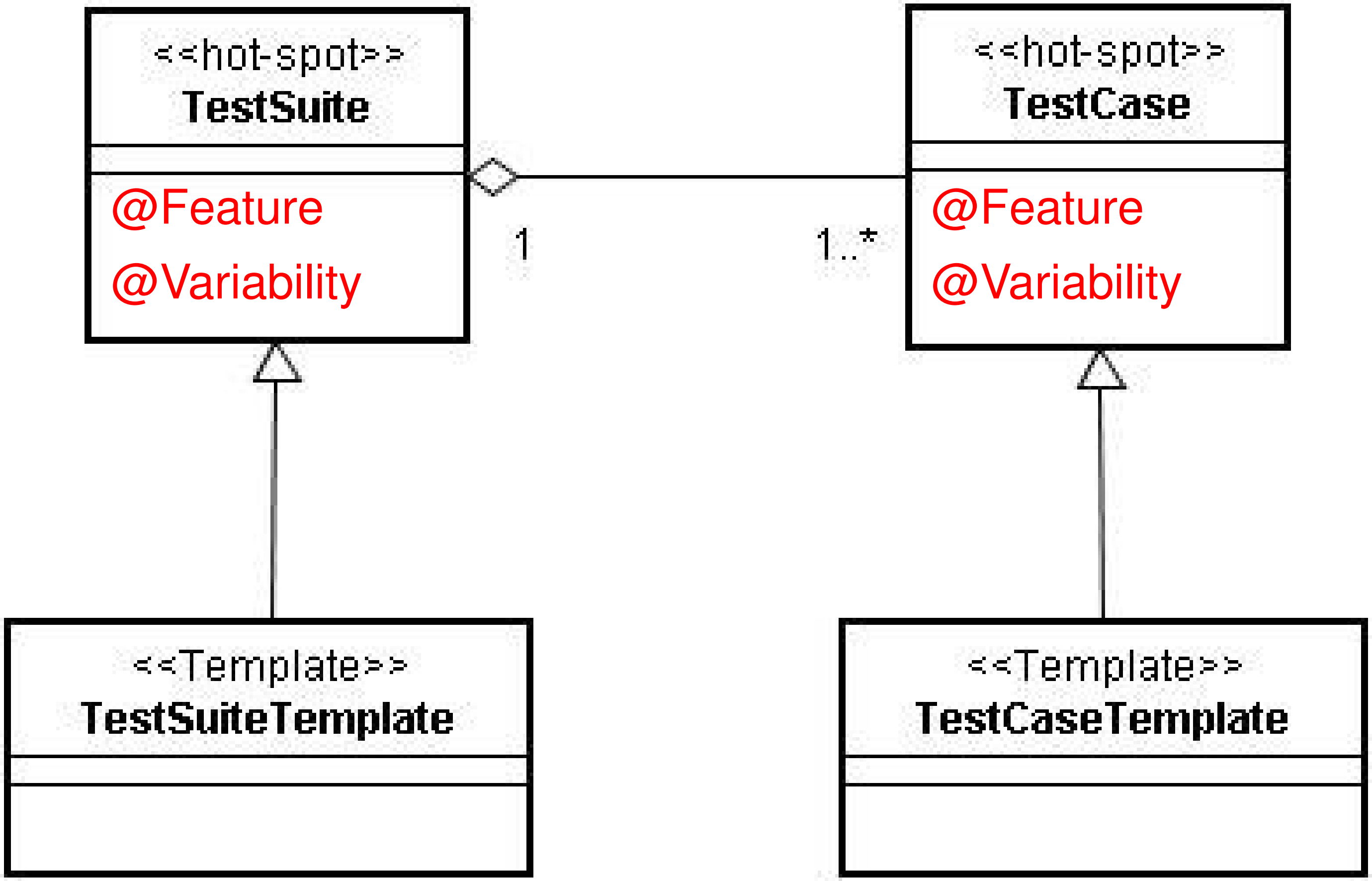
After

II. Generating and Refining the Approach Models



```
Configuration
  Component junit
    Component framework
      Template TestSuiteTemplate.xpt  (TestSuite)
      Template TestCaseTemplate.xpt  (TestCase)
    Component runner
      Component textui  (TXT)
      Component awtui  (AWT)
      Component swingui  (SWING)
    Component extensions
      Aspect RepeatedTestGeneric.aj  (Repeat)
      Aspect ActiveTestSuiteGeneric.aj  (Concurrent Execution)
      Template RepeatedTestGeneric.xpt  (Repeat)
      Template ActiveTestSuiteGeneric.xpt  (Concurrent Execution)
```

III. Implementing Variabilities with Templates



III. Implementing Variabilities with Templates

```
«IMPORT featuremodel»
«DEFINE Main FOR Feature»
    «FILE attribute + ".java"»
          package junit.framework;
          public class «attribute» extends TestSuite {
          }
    «ENDFILE»
«ENDDEFINE»
```

Before

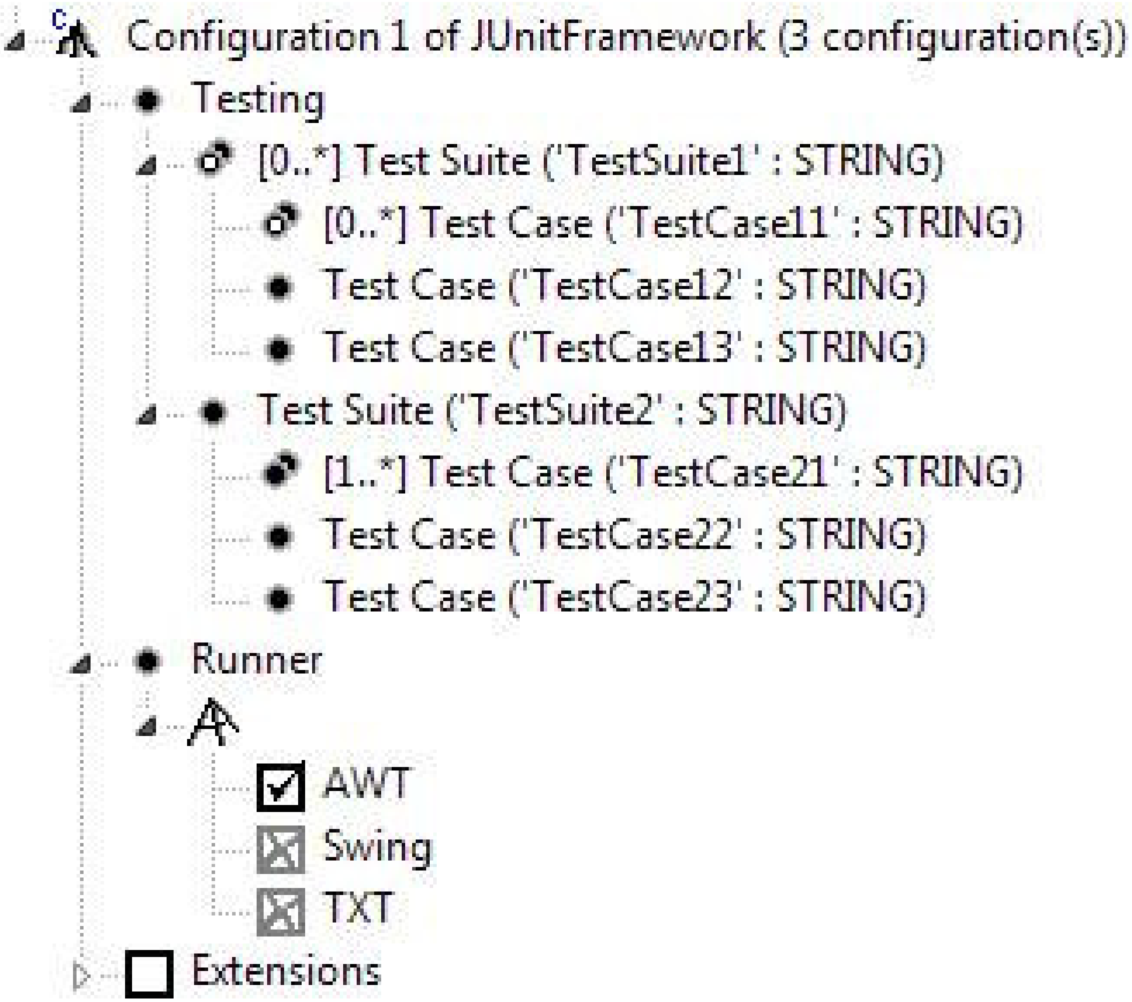
III. Implementing Variabilities with Templates

```
«IMPORT featuremodel»
«DEFINE Main FOR Feature»
   «FILE attribute + ".java"»
        package junit.framework;
        public class «attribute»  extends TestSuite {
                public static Test suite() {
                TestSuite suite = new TestSuite();
                «FOREACH features AS child»
                suite.addTestSuite(«child.attribute».class);
                «ENDFOREACH»
                return suite;
        }
   }
«ENDFILE»
«ENDDEFINE»
```

After

IV. Generating SPL Instances

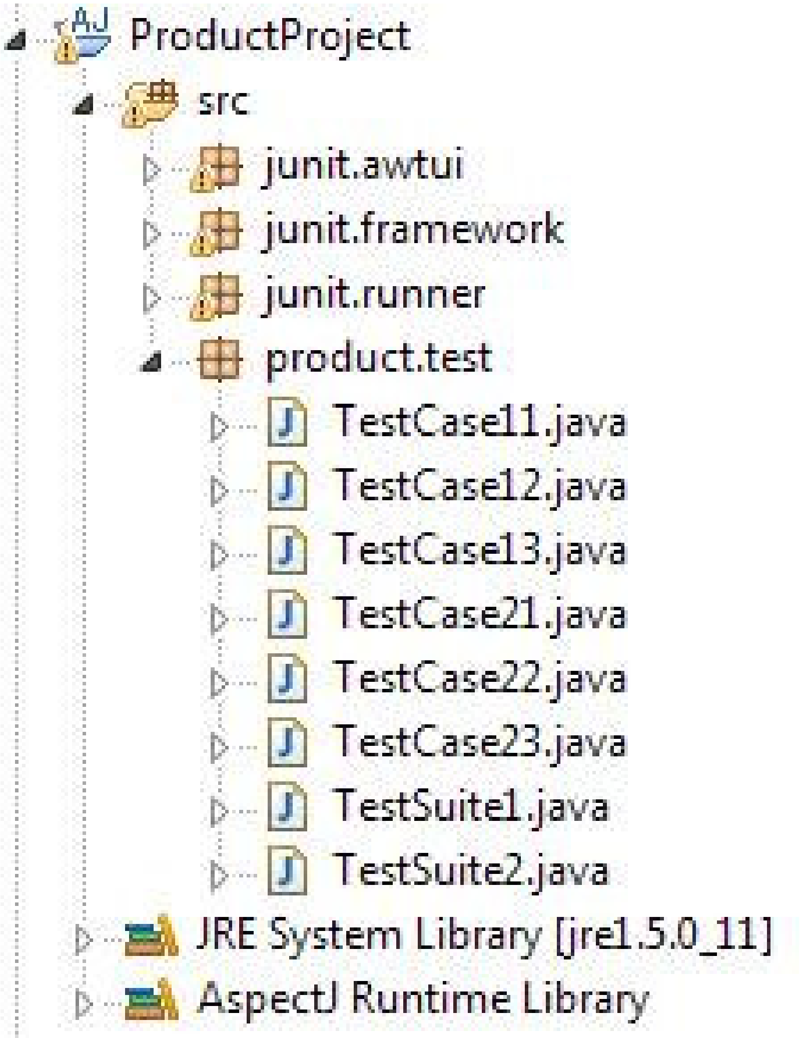
- Choose the Variable Features (Feature Model Instance)



- Configuration 1 of JUnitFramework (3 configuration(s))
  - ● Testing
    - [0..*] Test Suite ('TestSuite1' : STRING)
      - [0..*] Test Case ('TestCase11' : STRING)
      - ● Test Case ('TestCase12' : STRING)
      - ● Test Case ('TestCase13' : STRING)
    - ● Test Suite ('TestSuite2' : STRING)
      - [1..*] Test Case ('TestCase21' : STRING)
      - ● Test Case ('TestCase22' : STRING)
      - ● Test Case ('TestCase23' : STRING)
  - ● Runner
    - ☑ AWT
    - ☒ Swing
    - ☒ TXT
  - □ Extensions

IV. Generating SPL Instances

- Load the Product Code in a Eclipse Project



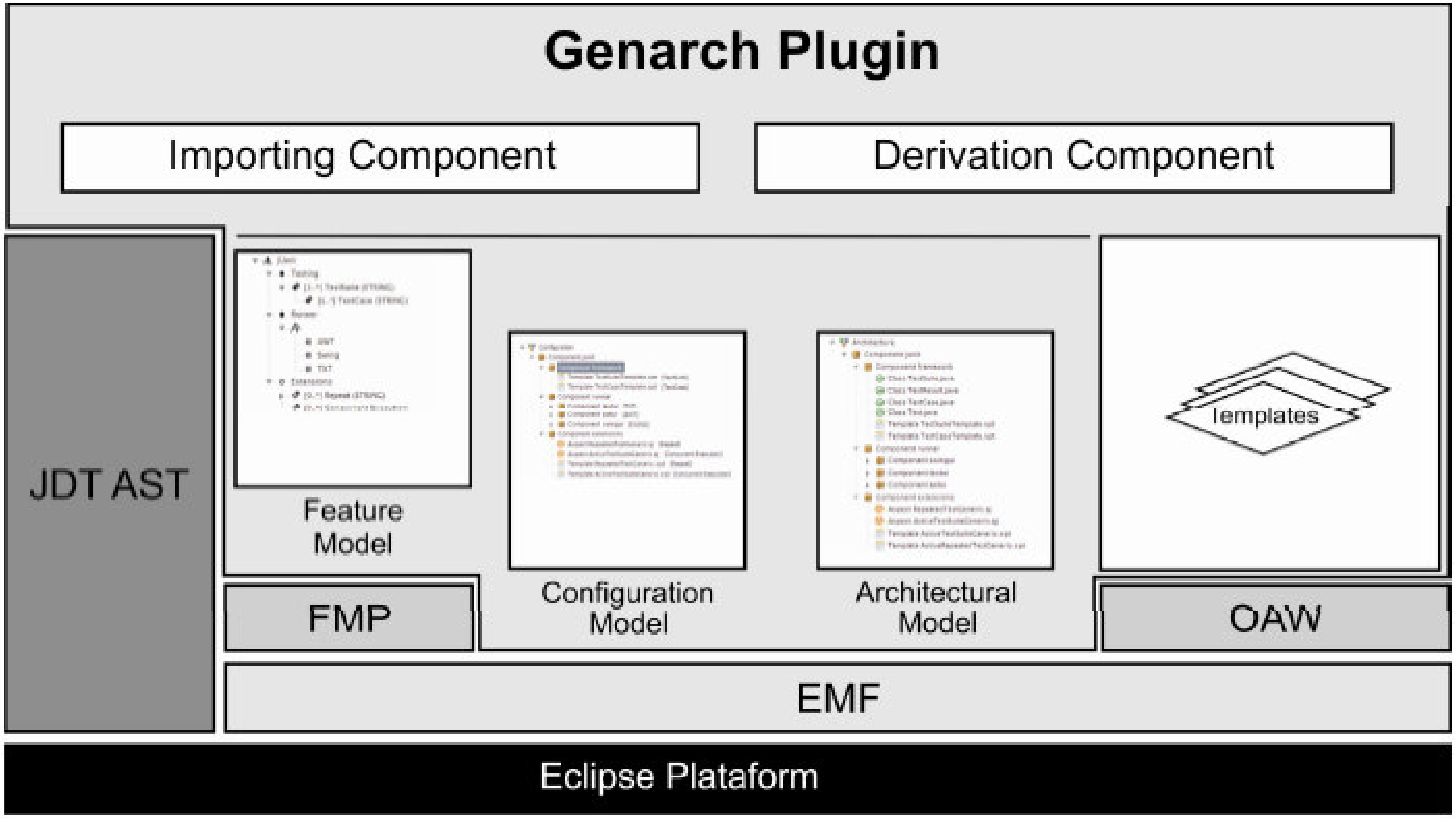
```
ProductProject
  src
    junit.awtui
    junit.framework
    junit.runner
    product.test
      TestCase11.java
      TestCase12.java
      TestCase13.java
      TestCase21.java
      TestCase22.java
      TestCase23.java
      TestSuite1.java
      TestSuite2.java
  JRE System Library [jre1.5.0_11]
  AspectJ Runtime Library
```

Agenda

- Introduction / Motivation
- Approach Overview
- Approach in Action
- Architecture Overview
- Discussions and Lessons Learned
- Conclusions and Future Work

© LES/PUC-Rio

Architecture Overview



Agenda

- Introduction / Motivation
- Approach Overview
- Approach in Action
- Architecture Overview
- Discussions and Lessons Learned
- Conclusions and Future Work

Synchronization between Annotations & Models

- In the current version, there is no available functionality to synchronize the SPL annotations and respective models.

- We are starting to work on the following functionalities:

  (i) removing of features which are not longer used by the configuration model or annotation;

  (ii) removing of mapping relationships in the configuration model which refer to non-existing features or implementation elements;

  (iii) removing of implementation elements from the architecture model which do not exist anymore;

  (iv) automatic creation of annotations in implementation elements based on information provided by the configuration model.

Integration with Refactoring Tools

- The integration of GenArch with existing refactoring tools involves several challenges, such as, for example:

  (i) to allow the creation of @Feature annotations to every refactoring that exposes or creates a new variable feature in order to present it in the SPL feature model to enable its automatic instantiation; and

  (ii) refactorings that introduce new extension points (such as, abstract classes or aspects or an interface) must be integrated with GenAch to allow the automatic insertion of @Variability annotations.

Architecture Model Specialization

- We are working on the definition of specializations of the architecture model.

- The specializations have the purpose to support other abstractions and mechanisms of specific technologies.

- The first specialization will support abstractions provided by the Spring framework, such as Spring beans, Spring aspects and their respective configuration files.

Agenda

- Introduction / Motivation
- Approach Overview
- Approach in Action
- Architecture Overview
- Discussions and Lessons Learned
- Conclusions and Future Work

Conclusions and Future Work

- Our tool combines the use of models and code annotations in order to enable the automatic product derivation of existing SPL architectures.
- The current version of GenArch will work as a base to provide a set of new and interesting SPL functionalities:
  - Customization of aspect libraries using feature models
  - Synchronization of Models
  - Composition with other different DSLs
  - Integration with refactoring tools
  - Specialization of the Architecture Model

Questions? Suggestions? Comments?

Laboratório de Engenharia de Software

GenArch (Generative Architectures): A Model-Based Product Derivation Tool

Elder Cirilo, Uirá Kulesza, Carlos Lucena

Software Engineering Laboratory
Computer Science Department
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)

Instantiation of Aspect Libraries

- Specification of features <<crosscutting >> and <<joinpoint>>.
- Specification of mapping between joinpoint features and concrete aspect joinpoints